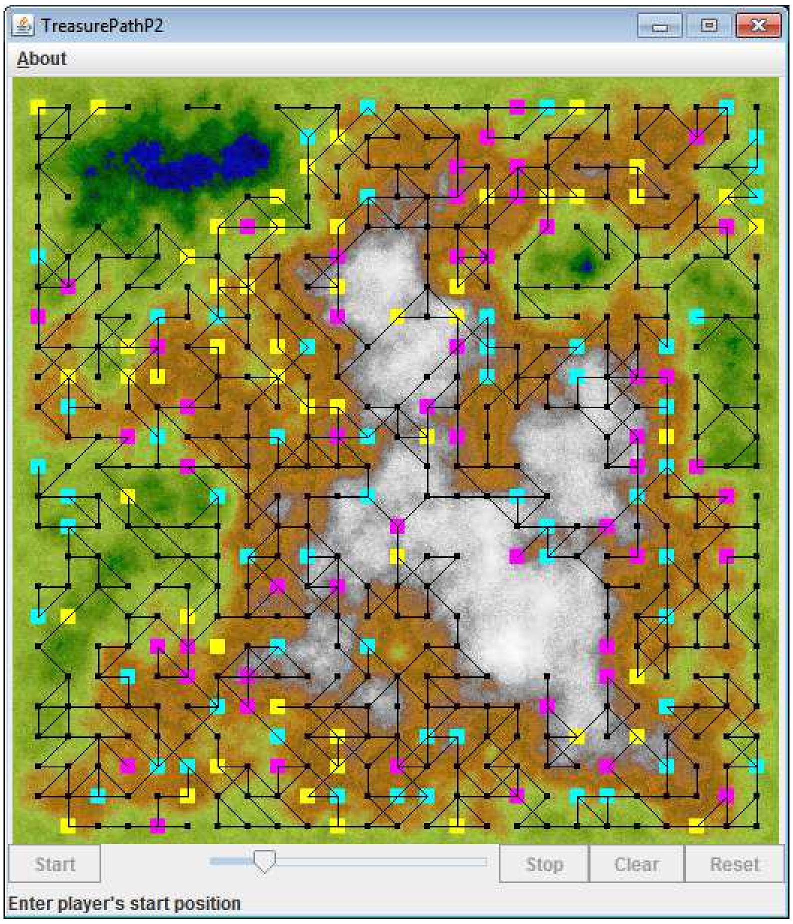
Comp 282 P2: Treasure Path due: 11 / 8 / 2015 Barnes Fall 2015

This is a graph project. Create a graph of traversable Waypoints generated from a terrain's data file (waypoint.txt) and drawn on terrain image (terrain282.png from P1). Again you must use the Simulation Framework for this project. I have described the project as having 3 phases as a way of suggesting an incremental design, develop and test plan. (This is only a suggestion, you can of course, make your design, develop, and test plan).



Phase 1 – Graph Generation

The first phase is very similar to the P1 assignment. I'll use the class name "Map" to refer to a graph class. Each Waypoint can have up to 8 neighbors; adjacent Waypoints that one could walk to from the current Waypoint. Not all possible adjacent Waypoints are navigable and not all adjacent Waypoints are neighbors. As in project 1 a Waypoint can be a place, a city, have a treasure, or have a map to a treasure. Use the same color and size coding from P1. Connectors should be drawn in black to show the path between neighbors.

For the first phase you need to design, implement, and test an application that will read a terrain data file and its associated terrain image file. It should create the Waypoints with neighbors that define the paths one could "walk" on the terrain. Your application should display the Waypoints and edges on the terrain image as described above. You can draw connectors between Waypoints twice (from A to B and from B to A) if you want. While the Java Collection Framework (JCF) does not have a graph or Waypoint class, you should use the JCF as much as possible in this project. As with project 1 use a generic HashMap to hold the Waypoints.

Phase 2 – Interactive Navigation

Your simulation of the A* path algorithm requires a start and goal Waypoint. Clicking on the map should select the closest Waypoint. The starting Waypoint and the goal Waypoint should both be shown with a blue Maker size of 4. A path from the starting Waypoint to the goal Waypoint is determined; if it exists. Not all pairs of Waypoints have connecting paths. If a path does not exist between Waypoints, that information should also be reported to the user. If a path does exist, the player (a red Bot) should traverse the path. This will be shown with a red path.

You should implement the A* algorithm presented in lecture, or a close variant. One of the objectives of this assignment is for you to gain experience with Collections (open and closed sets) and to use a JCF

PriorityQueue. Each Waypoint's neighbor has a cost (distance) to travel from the current Waypoint to the neighbor. You should visually show the determination of the Player's path. The path algorithm's set of open Waypoints (e.g., white Markers, size 3) as well as the closed (evaluated) Waypoints (e.g., gray Markers, size 2) should be displayed as Waypoints are added to each set. Each animation step represents the processing of a Waypoint in the open set. The different marker sizes should allow you to also see the history of the Waypoint with respect to path finding. The white (open set) and gray (closed set) Markers should be in the AnimatePanel's temporaryDrawable collection.

The Player class should have a wealth and strength value. The player starts with a strength of 2,000 and a wealth of 1,000. I may change the initial values for the player. As in P1, players lose strength equal to the distance travelled between Waypoints. If they travel through a city they can pay the city's cost for food out of wealth and gain that cost amount of strength. If they can't pay the total cost, they do not buy any food and they do not get any strength. If a player travels over a treasure they add the treasure's gold to their wealth, the value of that treasure becomes 0. The treasure's Waypoint's display is unchanged to reflect that it has, or once had, gold. All maps to that treasure are also unchanged. The player continues to their destination even when they have zero or negative strength or wealth. Report the player's strength and wealth at their goal. Your program should display (status line) and print (in the console) the following statements. These values reflect the step that just happened (after the step). This should help with debugging and grading. In the messages each "<value>" is replaced with actual numeric values.

```
Create HashMap <size> City <size> Gold <size> Map <size>
Start (<x>, <y>), Stop (<x>, <y>), Player <strength>  $ <wealth>
City (<x>, <y>) $ <cost>, Player <strength>  $ <wealth>
Gold (<x>, <y>) $ <gold>, Player <strength>  $ <wealth>
Map (<x>, <y>) Treasure (<x>, <y>) Player <strength> $ <wealth>
Path (<x>, <y>) to (<x>, <y>), length <n>
No path (<x>, <y>) to (<x>, <y>)
Success, goal (<x>, <y>) Player <strength> $ <wealth>
```

For example, when the path to the goal is initially determined report:

```
Path ( 20,  60) to (200, 200), length  13
```

Phase 3 – Treasure Maps.

In phase 3 after the player has determined a path and while they are walking the path they respond to treasure maps. When the player comes to a Waypoint with a treasure map they change their path to go to the treasure to get the gold. They again use the A* algorithm to determine the path from the current Waypoint to the treasure Waypoint. They remember their goal Waypoint. When the player gets to the treasure they gain its wealth in gold, set the gold value to 0, and they determine a new path from the current (treasure) Waypoint to their goal Waypoint. Note, the treasure could have been visited by the Player earlier and so have no gold. Player's can only have 1 treasure map at a time. So if they encounter a Waypoint with a treasure map while they have a treasure map, they ignore the new treasure map and proceed with their current path. When the player gets to the treasure they no longer have a treasure map.

The visual simulation of the A* path finding should show only the current path's open and closed sets. Consider a player walking a path. The current path's open and closed sets are visible. The player is walking on the current path. The player now "steps" on a Waypoint with a map. The current path is no longer valid, so it is no longer displayed. A new path to the map's treasure is determined. Once the map is determined the player walks on that path until they reach the treasure. This continues until the player reaches its goal.

Submission
Tentative grade ranges might be: phase 1 only 71 ... 73, phase 2 only 74 ... 87, phase 3 83 ... 100.

Submit your source files on Moodle as you did for P1. You do not need to submit any of the Simulation Framework classes (I have them). The names of all group members must be in a "header comment" at the start of every submitted java file. Submit “UML lite” class and state diagrams for your submission and an English description of your design and why you choose to use the JCF classes that you did use, and your design logic for the classes you wrote (no more than 5 pages). The names of all group members must be on the report. You can name your class files as you like. However, your “application class” that extends SimFrame must be named “<LastName(s)>P2.java” so that I have at least one file with the group member names in its name!

As with any journey or path, a good design and plan can make the trip more efficient…
Bon voyage!